AIAA–98–4807

PRELIMINARY RESULTS FROM THE APPLICATION OF AUTOMATED ADJOINT CODE GENERATION TO CFL3D

Alan Carle*
Rice University
Houston, Texas
carle@rice.edu

Mike Fagan†
Rice University
Houston, Texas
mfagan@rice.edu

Lawrence L. Green‡
NASA Langley Research Center
Hampton, Virginia
l.l.green@larc.nasa.gov

Abstract

This report describes preliminary results obtained using an automated adjoint code generator for Fortran to augment a widely-used computational fluid dynamics flow solver to compute derivatives. These preliminary results with this augmented code suggest that, even in its infancy, the automated adjoint code generator can accurately and efficiently deliver derivatives for use in transonic Euler-based aerodynamic shape optimization problems with hundreds to thousands of independent design variables.

Introduction

Automatic differentiation (AD) is a set of techniques for automatically augmenting computer codes to compute derivatives of their outputs with respect to their inputs. Numerous papers presented at recent Multidisciplinary Analysis and Optimization (MA&O) conferences have reported that AD can provide the derivatives required for use in simulation-based design.[1–8] These papers describe the use of ADIFOR, an AD tool for Fortran.[9,10] ADIFOR implements the *forward mode* of AD. For a code with n independent variables (or design variables) and m dependent variables, the forward mode computes the derivatives using time and space proportional to n. Obviously, for problems with a large number of independent variables, the computational cost of this method is prohibitive.

ADJIFOR, a substantially extended version of ADIFOR, implements the *reverse (or adjoint) mode* of AD. For a code with n independent variables and m dependent variables, the reverse mode computes the derivatives using time proportional to m, not n, albeit by using space proportional to the number of floating point operations required to execute the original code. Fortunately, as will be shown for the CFL3D (Computational Fluids Laboratory 3-Dimensional) code, special mathematical properties of steady-state solutions can be exploited to dramatically reduce the storage requirements of the reverse mode. Other reverse mode AD tools include Odyssée[11] and TAMC.[12]

The Shape Optimization Problem

Aerodynamic *shape analysis* requires a grid generator to be coupled with a flow solver. Given a set of *shape parameters*, the grid generator creates a grid. The newly created grid then becomes an input to the flow solver. The flow solver then computes the aerodynamic outputs. Often, the point of shape analysis is to determine the values of shape parameters that give rise to "favorable" aerodynamic outputs. The process of seeking shape inputs that lead to the favorable outputs is called *shape optimization*. Conversion of a shape analysis problem into a shape optimization

*Faculty Fellow, Member, AIAA, Department of Computational and Applied Mathematics

†Research Scientist, Department of Computer Science

‡Research Engineer, Senior Member, AIAA, Multidisciplinary Optimization Branch

problem requires defining exactly what criteria constitute "favorable." The definition of favorable includes an *objective function* to be minimized or maximized. Geometric and flow constraints might also be included.

Gradient-based shape optimization requires the derivatives of the objective function and the constraints with respect to the parameters that control the shape of the object. In the following text, Q represents the flow field, X the grid, and B the shape parameters. A grid generator, G, generates the grid X, given shape parameters B; that is, $X = G(B)$. The flow solver computes the final "converged" flow field by iterating a stepping function S. The stepping function computes the next iterate, using the current iterate and grid. This dependence is emphasized by writing $S(Q, X)$ for the step. The initial iterate in the procedure is indicated as Q_0, which is usually independent of the shape parameters B, except perhaps on the boundaries. The final, or converged, flow field is indicated as Q_*, where $Q_* = S(Q_*, X)$, meaning that the solution has reached a steady state. The objective function F is a function of both the flow field and the grid, and V is the value of F for a given Q and X; that is, $V = F(Q, X)$.

With this notation in place, the following pseudocode defines the canonical shape analysis procedure:

$X = G(B)$
$Q = Q_0$
Do until Q "is converged"
    $Q = S(Q, X)$
Enddo
$V = F(Q, X)$

As indicated previously, to solve gradient-based shape optimization problems it is necessary to compute the derivatives of the objective function and the constraints with respect to the shape parameters. To simplify the initial applications of ADJIFOR, no attempt has been made to compute derivatives for problems with constraints.[§] Hence for the simplified problems, only the derivatives of the dependent variable V with respect to the components of the independent variables B are required.

In this paper, a uniform notation for derivatives is used. The matrix representation for the derivative linear operator for any function Z, i.e., the Jacobian of Z, will be written J_Z. In addition, the derivative of any variable Z with respect to B will be written as Z', and the derivative of V with respect to a variable Z will be written as $\overline{Z}$. In summary, the ADJIFOR-generated code should compute V', or equivalently, $\overline{B}$.

For additional convenience, I is used for any identity matrix, 0 is used for the zero matrix, $|Z|$ is the number of elements in matrix Z, and Z^T is the transpose of matrix Z. The dimensions of matrices will either be obvious from context or explicitly indicated.

AD for Shape Optimization

The classic forward mode of automatic differentiation accumulates derivatives as a computation proceeds from the inputs to outputs. It follows the control flow of the original program and, for a matrix R with p columns, computes the matrix product $J * R$ to give the p directional derivatives with a time and space complexity that is roughly p times that of the original program. If $R = I$, the forward mode computes J. An alternative approach, the reverse mode, accumulates the derivatives in the opposite direction—from outputs to inputs. To propagate adjoints, one must be able to reverse the flow of the program, and record or recompute any intermediate value that nonlinearly affects the final result. Once these technical difficulties are overcome, then, for a matrix L with q rows, the matrix product $L * J$ can be computed with a time complexity that is roughly q times that of the original program. If $L = I$, the reverse mode computes J.

The need to record intermediate program values makes the storage requirements of adjoint codes potentially very high, particularly for iterative methods. Minimizing the storage requirements represents the most significant challenge to automatic adjoint tools. Checkpointing strategies[13] or additional mathematical knowledge can be used to reduce these requirements.

Assuming that sufficient storage for the reverse mode is available, the choice of forward mode or reverse mode for computing the Jacobian depends on the number of independent variables p, the number of dependent variables q, the ratio of the cost of computing a column of the Jacobian to the cost of computing the function using the forward mode O_f, and the ratio of the cost of computing a row of the Jacobian to the cost of computing the function using the reverse mode O_r. If $q * O_r < p * O_f$ then reverse mode is indicated. Since the operations performed by forward and reverse modes are vector operations, O_r and O_f actually depend on q and p, respectively. O_r and O_f depend on platform, compiler, and application as well. Typically, O_f for forward mode-based ADIFOR ranges from .5 to 4.0. In limited tests so far, O_r for reverse mode-based ADJIFOR ranges from 6.5 to 20. Hence, the reverse mode is particularly attractive for computing sensitivities for shape optimization problems with a large number of shape parameters, an objective function,

[§] The techniques described below apply directly to constraints that are functions of Q_* and X by simply expanding F into a vector-valued function that computes the objective function and the constraints. The dimension of the derivatives expands accordingly.

and a "few" flow constraints. For example, if $O_r = 20$ and $O_f = 2$, reverse mode outperforms forward mode whenever $p/q > 10$, where $q = 1 + \#\text{constraints}$.

To mathematically justify the approach to derivative computation for shape optimization problems, further details about the framework are elaborated. The framework is conceptually simple, relying only on the fact that derivatives are linear functions and, consequently, composition of derivatives is simple matrix multiplication. The derivatives are viewed as linear functions of all program variables in the following canonical order: shape parameters B, grid X, flow field Q, and objective value V. This convention is used solely for mathematical convenience. The implemented derivative computation does *not* form huge matrices and then multiply them together.

Using this framework, the derivative of the shape optimization problem can be written as

$$V' = L\ J_F\ J_{S_n}\ \ldots\ J_{S_1}\ J_G\ R,$$

where, using the notation for linear operators introduced above, J_Z represents the derivative of a function Z evaluated at its inputs. In particular, J_{S_k} is the Jacobian of S evaluated at Q_k, the flow field at step k. Moreover, J_{S_*} is the Jacobian of S evaluated at Q_*, the final flow field. Matrices L and R are block row and column projection matrices that select the desired independent and dependent variables, respectively.

For this problem, variable V is the desired output, and B is the vector of desired inputs, hence, the two projections are

$$L = \begin{pmatrix} 0_{1\times|B|} & 0_{1\times|X|} & 0_{1\times|Q|} & 1 \end{pmatrix}$$

and

$$R = \begin{pmatrix} I_{|B|\times|B|} \\ 0_{|X|\times|B|} \\ 0_{|Q|\times|B|} \\ 0_{1\times|B|} \end{pmatrix}.$$

As described previously, assuming sufficient storage for recording intermediate program values for each of the functions $G, S_1, \ldots, S_n$, and F, reverse mode computes V'. If the number of steps n is large, then a tremendous amount of storage will be required. Fortunately, it is possible to take advantage of mathematical properties of the flow solver to substantially reduce the storage requirements for a steady-state solution. This reverse mode variant is called the *iterated reverse mode*. Christianson provides mathematical justification of this approach when the flow solver is "sufficiently" converged.[14]

Applying the implicit function theorem to the steady-state condition

$$Q_* = S(Q_*, X)$$

gives

$$\begin{pmatrix} B' \\ X' \\ Q'_* \\ V' \end{pmatrix} = J_S \begin{pmatrix} B' \\ X' \\ Q'_* \\ V' \end{pmatrix}.$$

Since Q_* is a fixed point of S, the matrix $(B', X', Q'_*, V')^T$ is a fixed point of J_S. Recall that the framework expands derivative linear operators to cover *all* variables. Since S has no effect on B, X, or V, the appropriate entries in J_S will be 1.

When J_{S_*} is contractive (i.e., $\|J_{S_*}\| < 1$), the contractive mapping theorem guarantees that a fixed point of J_{S_*} can be computed by simple iteration. The flow solvers and test cases encountered so far appear to have the necessary contractive properties. Applying this technique to the shape optimization problem, V' can be computed as

$$V' = L\ J_F\ J_{S_*}\ \ldots\ J_{S_*}\ J_G\ R.$$

For this computation, note that the *same* operator J_{S_*} is used for each reverse-mode iteration. Note also that the number of iterations required for convergence of the derivatives is not necessarily the same as required for convergence of the original function. Consequently, instead of storing the intermediate values for all n steps, it is sufficient to store the values for a single step. Implementing the iterated reverse mode requires only two small changes (less than 10 lines of code) to the ADJIFOR-generated reverse-mode code: (1) turn off intermediate value recording for the first $n-1$ steps of S, saving the values only for step n, and (2) modify the control loop for S to repeatedly execute the reverse-mode code for S_* until the derivatives converge or until a user-specified number of iterations has been reached.

Description of the Codes

For the current study, G is a "home grown" wing grid generator named MYGRID, S is the stepping function in the CFL3D flow solver code, and the objective function F is taken to be the lift-to-drag ratio, which is obtained by dividing the computed lift coefficient, cl, by the computed drag coefficient, cd. The calculation of this objective function, cl/cd, was added to the original CFL3D code. Description of MYGRID and CFL3D follows.

MYGRID Grid Generation Code

MYGRID implements a fast and simple algebraic method for generating wing grids. The grid generation code was developed (in Fortran) for use in ADIFOR and ADJIFOR studies. It is very robust in grid generation, but the code does not include many of the

advanced techniques commonly used in commercial grid generation packages to ensure high quality grids.

MYGRID defines 3-D wings by a set of wing sections, using an expanded definition of the NACA four digit airfoil section family, based upon real numbers for the maximum thickness, maximum camber, and location of maximum camber, rather than the usual integer designation. This allows for airfoil shapes to be incrementally perturbed in a continuous, rather than a discrete, fashion, thus enabling the construction of accurate finite-difference approximations to verify the results of ADIFOR and ADJIFOR. Each wing section is described by eight design parameters: *xle*, *yle*, and *zle* (the x, y and z leading edge coordinates), *crd* (the wing section chord length to trailing edge), *cmx* (the maximum camber line height in y-direction), *xcm* (the streamwise location of maximum camber height), *thk* (the streamwise maximum airfoil thickness), and *tws* (the section twist angle).

The number of design variables can be increased by simply increasing the number of wing sections that are specified. The code also allows the user to specify the number of grid points in each of the coordinate directions and provides a few choices that affect the grid stretching and distribution.

The MYGRID code produces single-block grids that were used in the initial demonstrations of this adjoint technology. Subsequently, a utility program was developed by Biedron of NASA Langley Research Center, which splits a single-block grid and its associated CFL3D input file into a user-specified number of subset grid blocks, while also splitting the boundary condition specification within the associated single-block CFL3D input file into a multiblock input specification. This grid block and input file splitting utility program was used within the current work to provide a mechanism to decompose a large grid into many smaller pieces for parallel processing.

The CFL3D Flow Solver Code

The CFL3D code is a general purpose computational fluid dynamics (CFD) solver developed by Thomas, Rumsey, and Biedron of the NASA Langley Research Center, with contributions from numerous other researchers. From its inception in the early 1980's, the CFL3D code has been continuously improved, applied to a wide variety of problems, verified extensively by experiment and other CFD results, and widely distributed for use in industry.[15–22]

The CFL3D code solves the time-dependent Reynolds-averaged Navier-Stokes equations in conservation form using upwind-biasing for the convective and pressure terms, and central differencing for the shear stress and heat transfer terms. The code includes the ability to solve inviscid, laminar, or turbulent flows around complex 2-D or 3-D geometries using one of four possible grid schemes (point-matched, patched, overlapped, or embedded). CFL3D includes the ability to compute steady or unsteady flows with implicit time advancement. Both multigrid and mesh sequencing techniques can be used for convergence acceleration. The code also provides numerous turbulence models, including Baldwin-Lomax, Baldwin-Barth, and Spalart-Allmaras, as well as several other popular models.

Two distinct versions of the CFL3D code were used in this work: (1) the sequential CFL3D version 5.0 code and (2) an existing block-parallel version of CFL3D, from the version 4.1 code that employs MPI (Message Passing Interface) routines to implement a distributed-memory parallel flow solution.[23] Although these two codes are substantially different internally, and the sequential version 5.0 code includes some new features not found in the parallel version 4.1 code, they have been found to produce essentially the same results for test cases similar to those used in this work. A forthcoming NASA Technical Memorandum documents the CFL3D version 5.0 code and its differences from previous code versions, including: (1) sliding patched-zones for use in rotor-stator computations and (2) improved computational efficiency and memory usage.

For simplicity, only a limited subset of the possible CFL3D code options have been demonstrated within the scope of the work presented in this paper, although the only portions of the CFL3D code that were purposely avoided were the turbulence models. Based upon previous work with the ADIFOR code generation tool, the ADJIFOR-generated CFL3D code is expected to work correctly and accurately for the entire suite of boundary conditions, as well as the numerous solver and multigrid options within the code. It is expected that the primary impact of these additional options will be to increase the storage requirements of the adjoint code in proportion to the increased complexity of the computation. Adjoint code for the turbulence models could easily be incorporated in the adjoint code. Also, the use of patched or overset grids would require ADJIFOR processing of, respectively, the RONNIE or MAGGIE utility programs provided with CFL3D; ADJIFOR processing of these utilities has not yet been attempted.

For this current work, the CFL3D code was used to solve steady, inviscid, transonic flow around a simple 3-D transport wing. The algorithmic choices included the use of local time stepping, Roe's flux-difference splitting scheme, and scalar tri-diagonal matrix inversion with smooth flux-limiters. Multigrid and grid

sequencing were not used in these demonstrations. The test cases used either single-block grids, or grids that had been decomposed into smaller, point-matched PLOT3D multiblock style grids via the splitter utility program previous described.

Description of the Test Problem

The choice of a test problem for this ADJIFOR demonstration was influenced by two considerations: (1) the problem needed to resemble a realistic transonic shape optimization problem, and (2) the problem needed to be small enough to permit validation of numerical results using ADIFOR and finite differences. In light of these considerations, a small-sized shape parameter specification was developed. Specifically, the test problem uses 88 shape parameters to define a swept and tapered wing, similar to those used on numerous commercial transport aircraft today. Given these 88 shape parameters, MYGRID was used to generate $33 \times 9 \times 9$ and $65 \times 17 \times 17$ one-zone grids. The grid splitter was then used to split each of these one-zone grids into two-zone, four-zone and eight-zone grids. Test cases were run under an Euler regime at Mach .84, alpha 3.06°. CFL3D 5.0 was tested only on the $33 \times 9 \times 9$ one-zone grid. CFL3D 4.1 was tested on all eight grids. These grid sizes was chosen for this adjoint demonstration as a compromise between the flow modeling resolution and the initial in-core storage requirement for the adjoint flow solver. The adjoint capability was first demonstrated on the flow solver, without using the grid generation package. Each of the grid x-y-z coordinates input to CFL3D was considered to be an independent variable. This yielded a potentially large number of design variables, up to a total of 8019 ($33\times9\times9\times3$), with minimal time and storage requirements for the CFD solver.

Currently, adjoint versions of the grid and the flow solver have been coupled to produce flow sensitivities with respect to the shape specification parameters (*xle*, *yle*, *zle*, *crd*, *cmx*, *xcm*, *thk*, and *tws*) at each of the input sections. The number of independent design variables can be increased by simply increasing the number and distribution of input wing sections, so long as the spanwise grid resolution is approximately the same as the number and distribution of input sections. For this test case, there are 11 input wing sections and 8 design variables per wing section for a total of 88 shape specification design variables. If many more (or more densely spaced) wing sections are specified than spanwise grid lines computed, the process will still work, but some of the design variables may become ineffective, due to the linear interpolation between the input wing sections; several interpolations to obtain grid lines may be possible, and the effect of a design variable at any one input section then becomes less clear.

For this test case, the wing span is taken to be 1.0, the root chord is 0.6737, the tip chord is 0.3789. For simplicity of grid generation, a NACA 0010 wing section was used. The grid generation for this case was done with the MYGRID program, described previously. Since multigrid was not used in this study, converging the flow solution for this problem took about 1000 cycles.

Since the grid is so coarse, particular attention was given to grid stretching and distribution in order to obtain the most reasonable inviscid flow solution grid possible within the limitations of MYGRID. In the streamwise coordinate direction with 33 (or 65) grid points, the grid direction index starts, as is common, at the lower downstream wake, wraps around the airfoil from the lower trailing edge, to the leading edge, to the upper trailing edge, and continues to the upper downstream wake. Grid clustering has been provided near the leading and trailing edges for each wing section. An unusually large number of grid cells (50 percent) were placed in the wake, distributed equally between the upper and lower wake regions, to improve the grid distribution for this coarse grid case. Both the wing normal and spanwise directions have 9 (or 17) points. In the normal direction, grid points are somewhat clustered toward the airfoil surface; grid lines are perpendicular to the airfoil surface near the leading edge, but vertical beyond the airfoil maximum thickness point and in the wake. In the spanwise direction, the grid points are equally spaced along the wing span. In the test problem, the wing tip has zero thickness and the outer boundary is placed at a distance of 5 wing semi-spans away from the airfoil surface.

It is worth noting that, from the authors' experience, the initial ADIFOR/ADJIFOR validation for accuracy, relative to carefully constructed finite-difference approximations, can usually be done on such coarse grids as described above. In the current studies, grids as coarse as $17\times5\times5$ points were used in the initial verification of ADIFOR and ADJIFOR results. The accuracy of the ADIFOR/ADJIFOR derivatives for reasonably short runs, relative to such finite-difference approximations, appears to be independent of the resolution of the flow field features. This is true as long as the coarseness of the grid does not lead to inconsistencies in the basic flow solver algorithm. In fact, although the $17\times5\times5$ grid is already extremely coarse, it may be possible to do these validation tests on a $9\times3\times3$ grid, but the authors were uncertain whether that level of coarseness would violate some unknown 5-point operator assumption that may be buried deep within the CFL3D computational algorithm. There is

probably no way to prove that validation can always be done on these coarse grids; it may even be possible to prove that such tests can produce misleading, or incorrect, results for some cases. However, the virtue of such coarse grid validation, if it works, is that the results can be converged quickly to machine zero with limited time invested in grid generation.

Results

The goal of this work was to investigate the effectiveness of the ADJIFOR-generated adjoint code, in conjunction with the iterated reverse mode. Computing platforms used in this effort included: an 8-processor Sun Enterprise E4000 shared memory server, a 4-processor IBM SP2 shared memory node, and a 4-processor SGI Power Challenge shared memory node. In the following text and tables, these three shared memory processors are referred to as the SUN SMP, IBM SMP and SGI SMP, respectively. For CFL3D 5.0, test cases were executed using a single processor, and for CFL3D 4.1, two processors (one host and one compute node) were used for the one-zone test cases, three processors (one host and two compute nodes) for the two-zone test cases, five processors (one host and four compute nodes) for the four-zone cases, and nine processors (one host and eight compute nodes) for the eight-zone cases.[¶] All test cases were run using 64-bit floating-point arithmetic.

Table 1 shows derivative values for $\overline{tws}$ computed using the forward mode by ADIFOR and the iterated reverse mode by ADJIFOR for the $33 \times 9 \times 9$ one-zone test case using the sequential and parallel versions of CFL3D. For forward mode, the ADIFOR-enhanced MYGRID and 1000 steps of the ADIFOR-enhanced CFL3D were executed. For iterated reverse-mode, the ADJIFOR-enhanced MYGRID and 1000 steps of CFL3D, followed by 1000 adjoint iterations of CFL3D were executed. Forward-mode and iterated reverse-mode derivative values for CFL3D 5.0 show excellent agreement. The forward-mode and iterated reverse-mode derivative values for CFL3D 4.1 likewise show excellent agreement. For both codes, finite-difference approximations (not shown in the table) agree with the ADIFOR-generated derivatives to more than six significant figures. The discrepancies between the derivatives computed for CFL3D 5.0 and CFL3D 4.1 have not yet been investigated.

Table 2 presents timings based on CFL3D 5.0, including the shape analysis procedure, finite differences, ADIFOR-generated forward-mode derivative code, and the ADJIFOR-generated adjoint code using the same $33 \times 9 \times 9$ one-zone test case. Notice that the cost of computing all 88 derivatives using ADJIFOR on the SUN SMP is about 10 percent of the cost of using finite differences. The timing numbers shown in Table 2 is intended to illustrate the performance of ADJIFOR-generated code relative to other standard methods for computing derivatives. As such, they do not reflect the possible use of coarse-grained parallelization within finite differences or forward-mode derivative computations to improve performance. Also, the tables do not reflect the potential for improved efficiency of ADIFOR-generated code through the use of the incremental iterative method developed by Taylor and Oloso.[24]

Tables 3 and 4 summarize the time and storage requirements of the iterated reverse mode version of CFL3D 4.1. In Table 3, for each of the eight test cases, the following information is provided: (1) time required for the original function, (2) time required for the iterated reverse mode, (3) the ratio of the iterated reverse mode and function times. Table 3 shows that the iterated reverse mode is scaling better than the original CFL3D 4.1 function evaluation as the number of grid zones and processes is increased. The iterated reverse mode shows excellent performance on all eight test cases.

In Table 4, for each of the eight cases, the following information is provided (per compute node): (1) memory required for the original function, (2) memory (static and dynamic) for the iterated reverse mode, (3) disk space required for the iterated reverse mode, and (4) bytes per point summaries of the dynamic memory and disk requirements of the iterated reverse mode. The table indicates an approximation of 8000 bytes of dynamic memory per grid point and 32000 bytes of disk space per grid point for an iterated reverse mode calculation. This approximation yields a 3.2 GByte dynamic memory requirement and a 12.8 GByte disk space requirement for the target 400,000 point grid. Consequently, a 32-processor parallel computer consisting of processors with 128 MBytes memory and 400 Mbytes disk could be used to compute the derivatives for this 400,000 point problem.

Finally, in Fig. 1, the convergence behavior for $\overline{xle}$ for 6 of the 11 wing sections is shown for CFL3D 5.0. Convergence for all other design parameters and wing sections is similar. Derivatives do indeed converge, and appear to do so in perhaps half as many steps as the original flow solver.

It should be noted that several successful "by-hand" (rather than automated) adjoint demonstrations with CFD codes already exist.[25–28] The variations in these adjoint solution techniques and the iterated reverse

[¶] Executing CFL3D 5.0 on a single processor of multiprocessor hardware enabled the fairest possible performance comparisons between the various sensitivity-enhanced sequential and parallel versions of CFL3D.

mode make direct comparisons difficult. It is expected that ADJIFOR-generated code will always be more pessimistic in data storage requirements than the best by-hand adjoint methods employing significant knowledge about the code structure and solution process. Nonetheless, the performance of the ADJIFOR-generated code is competitive with the by-hand methods. Furthermore, the time required to produce adjoint versions of existing codes using ADJIFOR is surely orders of magnitude less than the time required for similar by-hand implementations.

Significance of Results

First, this work demonstrated the use of the ADJIFOR automatic adjoint code generation tool for the computation of shape sensitivities for a small problem. The derivatives for the design variables in this small problem have been verified to be accurate, thus providing confidence that the ADJIFOR tool is working correctly. This is significant because the validation of the adjoint results by finite differences or forward-mode differentiation will become more cumbersome as the number of design variables increases, and as the adjoint code becomes more efficient relative to the comparison methods.

Second, this work developed the iterated reverse mode for CFL3D. The iterated reverse mode uses a well-converged solution to store intermediate program values and then repeatedly executes the final iteration until the derivatives converge. This technique provides accurate derivatives with substantially smaller storage requirements than the standard reverse mode, and, in addition, appears to converge the derivatives in fewer iterations than were required for the solution convergence. The reduced storage requirements should make it possible to tackle much larger sensitivity and optimization problems.

Third, this work demonstrated that the ADJIFOR prototype is capable of providing shape sensitivities at a cost comparable to about 7 evaluations of CFL3D, on the SUN SMP, for a test case with 88 design variables. This compares favorably to the 89 function evaluations that would have been required for one-sided finite differences. These timing results agree with expectations.

The current demonstration represents a significant advance in the ability to automatically generate sensitivity code for sensitivity analysis or shape optimization using a widely distributed industrial-grade aerodynamic solver. Furthermore, a special effort has been made to accommodate the aerospace industry's need for rigorous validation. More work remains to be done on several fronts to make this demonstration more realistic.

Conclusions

The ADJIFOR automatic adjoint code generation tool has been applied to both a sequential and a parallel version of the CFL3D computational fluid dynamics code. The resulting ADJIFOR-generated codes have been demonstrated with one-zone, two-zone, four-zone, and eight-zone grids. The ADJIFOR application to these CFL3D code versions produced exact derivatives, with respect to shape parameters, of a sample objective function: the lift-to-drag ratio. The computed reverse-mode derivatives were compared with both forward-mode results and finite-difference approximations to validate their accuracy. The derivatives were obtained for steady-state problems using a technique known as the iterated reverse mode which records the last step in a function convergence and replays this information during the adjoint solution process until the derivatives converge.

The resulting adjoint sensitivity analysis was executed on multiprocessor parallel computers using the amount of storage typically available on these kinds of machines. For 88 design variables, the reverse mode code required execution times ranging from 7 to 21 function evaluations, depending upon machine type and compiler options. For this example, the reverse-mode derivatives converge to acceptable accuracy bounds within about half the number of iterations required for the function itself to converge to a steady state, resulting in an additional performance benefit relative to either the forward-mode or finite-difference methods for computing derivatives.

Remaining Work

The most immediate issue remaining for the ADJIFOR prototype is the reduction of the storage required for intermediate values in ADJIFOR-generated code. Two strategies will be investigated: eliminating the storage for unnecessary intermediate values (such as those that are only used linearly), and recomputing, rather than storing, other intermediate values.

Additional improvements to the prototype will be driven by a more thorough investigation of the adjoint code generated for CFL3D. The ADJIFOR-generated code will be demonstrated for laminar and turbulent flows, with other boundary condition options, and on larger, multiblock grids with grid sequencing, multigrid, and other convergence acceleration techniques. Previous experience with the forward-mode ADIFOR tool suggests that none of these issues should pose a difficult problem for the reverse-mode ADJIFOR tool. Each issue, however, must be addressed with a view toward rigorous validation for accuracy and the best-attainable efficiency. Also, the efficiency of the adjoint code must be demonstrated within a realistic opti-

mization problem, rather than just within a sensitivity analysis.

From discussions with engineers doing shape optimization for a major aerospace firm in the United States, the authors believe that the minimum realistic target test problem for this adjoint demonstration is a wing-body configuration in inviscid flow with at least 400,000 grid points and about 500 design variables. In such a test problem the grid size, the configuration complexity, and the number of design problems would be consistent with the current practices for shape optimization of new aircraft.

It is believed that grid size can be increased with little or no performance penalty using the ADJIFOR-generated parallel version of CFL3D, which uses MPI message passing to distribute large problems across numerous processors working in parallel. Thus, each processor is required to solve only a small piece of a larger problem. In theory, parallelism provides access to the large amounts of storage space required for the iterated reverse mode. It has not yet been shown, however, that good flow-solver convergence, a prerequisite for using the iterated reverse mode, can be achieved for grids that have been divided into a large number of zones.

One question that has only partially been answered at this point is how the convergence of function and adjoint codes in the shape analysis procedure affects the values of derivatives computed by the iterated reverse mode. In this work, the derivatives were observed to converge to acceptable accuracy bounds in fewer iterations than the function required to reach a steady state, starting the adjoint calculation from a well-converged function solution. It is not known whether this will be true for a broad range of configurations and classes of shape parameters. Also, the exact mechanism for this rapid convergence of the derivatives is not well understood. Convergence of the shape analysis procedure based, as is usual, on the convergence of the configuration force and moment coefficients (usually 3 or 4 orders of magnitude convergence) rather than on the convergence of the flow field itself (6 or more orders of magnitude convergence), will impact the accuracy of the derivatives within the shape sensitivity analysis and shape optimization procedures. But the exact impact of such reduced convergence has not been quantified. Furthermore, it is not known whether the adjoint method demonstrated here can be applied to time-dependent problems. In fact, it is expected that many, if not all, function iterations must be logged in order to construct accurate reverse-mode derivatives for time-dependent problems. The impact of logging many iterations of a time-dependent problem will be a huge increase in the storage requirements necessary to solve the adjoint problem.

Finally, there is theoretical and practical interest in clarifying the relationship between the iterated reverse mode and the various by-hand adjoint approaches. Ideally, this investigation will suggest ways in which automated methods may be improved to achieve efficiency comparable to the best by-hand approaches.

Acknowledgements

The authors wish to express their sincere thanks to Dr. Tom Zang, head of the MultiDisciplinary Optimization Branch (MDOB) at NASA Langley Research Center (LaRC), without whose support through both funding and advocacy this project would not have been possible. The authors wish to thank Dr. Chris Rumsey of the Aerodynamic and Acoustic Methods Branch (AAMB) at NASA LaRC for the guidance he provided to the authors in the use of the CFL3D code. The authors also wish to express their thanks to Dr. Bob Biedron of AAMB at NASA LaRC for his validation efforts with the parallel CFL3D code version and the development of the grid block splitter code for the adjoint solution of large grid problems. The authors express their thanks to Dr. Perry Newman of MDOB at NASA LaRC for his direction and support of MDOB research in automatic differentiation tools. The authors express their thanks to Drs. Shreekant Agrawal, Geojoe Kuruvila, Peter Hartwich, Pichuraman Sundaram, James Hager, Bob Narducci, and Eric Unger of Boeing Long Beach for their informative discussions with the authors about the status of shape optimization within the NASA High Speed Research Program. Finally, the authors express their thanks to Dr. Chris Bischof of Argonne National Laboratories for mathematical developments contributing to the ADJIFOR tool.

This work was supported by the National Aeronautics and Space Administration under Cooperative Agreement Number NCC 1 234, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008. Carle and Fagan were responsible for adjoint code generation and initial code execution demonstrations. Green was responsible for the problem definition, grid generation, and CFL3D code support.

References

[1]Unger, E., and Hall, L., "The Use of Automatic Differentiation in an Aircraft Design Problem," *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-94-4260-CP, Panama City, FL., Sept. 1994, pp. 64–72.

[2]Bischof, C., Knauff, T., Green, L., and Haigler, K., "Parallel Calculation of Sensitivity Derivatives for

Aircraft Design Using Automatic Differentiation," *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-94-4261-CP, Panama City, FL., Sept. 1994, pp. 73–86.

[3]Korivi, V., Sherman, L., Taylor, A., Hou, G., Green, L., and Newman, P., "First- and Second-order Aerodynamic Sensitivity Derivatves via Automatic Differentiation with Incremental Iterative Methods," *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-94-4262-CP, Panama City, FL., Sept. 1994, pp. 87–120.

[4]Korivi, V., Taylor, A., and Newman, P., "Aerodynamic Optimization Studies using a 3-d Supersonic Euler Code with Efficient Calculation of Sensitivity Derivatives," *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-94-4270-CP, Panama City, FL., Sept. 1994, pp. 170–194.

[5]Su, J. and Renaud, J., "Automatic Differentiation in Robust Optimization," *6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-96-4005-CP, Bellevue, WA., Sept. 1996, pp. 201–215.

[6]Wujek, B., and Renaud, J., "Automatic Differentiation for More Efficient Multidisciplinary Design Analysis and Optimization," *6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-96-4117-CP, Bellevue, WA., Sept. 1996, pp. 1151–1166.

[7]Moen, C., Spence, P., Meza, J., and Plantenga, T., "Automatic Differentiation for Gradient-based Optimization of Radiatively Heated Microelectronics Manufacturing Equipment," *6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Ooptimization*, AIAA-96-4118-CP, Bellevue, WA., Sept. 1996, pp. 1167–1175.

[8]Issac, J., and Kapania, R., "Aeroelastic Sensitivity Analysis of Wings using Automatic Differentiation," *6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-96-4119-CP, Bellevue, WA., Sept. 1996, pp. 1176–1186.

[9]Bischof, C., Carle, A., Corliss, G., and Griewank, A., "ADIFOR—Generating Derivative Codes from FORTRAN Programs," *Scientific Programming*, Vol. 1, 1992, pp. 11–29.

[10]Bischof, C., Carle, A., Khademi, P., and Mauer, A., "Adifor 2.0: Automatic Differentiation of Fortran 77 Programs," *IEEE Computational Science and Engineering*, Vol. 3, No. 3, Fall 1996, pp. 18–32.

[11]Rostaing, N., Dalmas, S., and Galligo, A., "Automatic Differentiation in Odyssée," Tellus, 45A, 1993.

[12]Giering, R., and Kaminski, T., "Recipes for Adjoint Code Construction," ACM TOMS, 1998, in press.

[13]Griewank, A., "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation," *Optimization Methods and Software*, Vol. 1, No. 1, 1992, pp. 35–54.

[14]Christianson, B., "Reverse Accumulation and Attractive Fixed Points," *Optimization Methods and Software*, Vol. 3, 1994, pp. 311–326.

[15]Biedron, R., and Thomas, J., "A Generalized Patched-Grid Algorithm with Application to the F-18 Forebody with Actuated Control Strake," *Computing Systems in Engineering*, Vol. 1, No. 2–4, 1990, pp. 563–576.

[16]Compton, W., Thomas, J., Abeyounis, W., and Mason, M., "Transonic Navier-Stokes Solutions of Three-Dimensional Afterbody Flows," NASA TM 4111, July 1989.

[17]Ghaffari, F., Luckring, J., Thomas, J., Bates, B., and Biedron, R., "Multiblock Navier-Stokes Solutions About the F/A-18 Wing-LEX-Fuselage Configuration," *Journal of Aircraft*, Vol. 30, No. 3, 1993, pp. 293–303.

[18]Rumsey, C., Biedron, R., and Thomas, J., "CFL3D: Its History and Some Recent Applications," NASA TM 112861, May 1997, presented at the "Godunov's Method for Gas Dynamics" Symposium, Ann Arbor, MI, May 1997.

[19]Rumsey, C., and Vatsa, V., "Comparison of the Predictive Capabilities of Several Turbulence Models," *Journal of Aircraft*, Vol. 32, No. 3, 1995, pp. 510–514.

[20]Rumsey, C., Sanetrik, M., Biedron, R., Melson, N., and Parlette, E., "Efficiency and Accuracy of Time-Accurate Turbulent Navier-Stokes Computations," *Computers & Fluids*, Vol. 25, No. 2, 1996, pp. 217–236.

[21]Thomas, J., Krist, S., and Anderson, W., "Navier-Stokes Computations of Vortical Flows Over Low-Aspect-Ratio Wings," *AIAA Journal*, Vol. 28, No. 2, 1990, pp. 205–212.

[22]Vatsa, V., Thomas, J., and Wedan, B., "Navier-Stokes Computations of a Prolate Spheroid at Angle of Attack," *Journal of Aircraft*, Vol. 26, No. 11, 1989, pp. 986–993.

[23]Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., and Dongarra, J., *MPI: The Complete Reference*, MIT Press, 1995.

[24]Taylor, A. C., III, "Automatic Differentiation of Advanced Flow-Analysis Codes in Incremental Iterative Form for Multidisciplinary Applications," Old Dominion University Research Foundation (ODURF), Tech. Rep. 96–147, 1996.

[25]Reuther, J., Alonso, J. J., Rimlinger, M. J., and Jameson, A., "Aerodynamic Shape Optimization of Supersonic Aircraft Configurations via an Adjoint Formulation on Distributed Memory Parallel Computers," AIAA Paper No. 96-4045, Sept. 1996.

[26]Anderson, W. Kyle, and Venkatakrishnan, V., "Aerodynamic Design Optimization on Unstructured Grids with a Continuous Adjoint Formulation," AIAA Paper No. 97-0643, 1997.

[27]Anderson, W. Kyle, and Bonhaus, Daryl, L., "Aerodynamic Design on Unstructured Grids for Turbulent Flows," NASA TM 112867, Jun. 1997.

[28]Kuruvila, G., Hager, J. O., and Sundaram, P., "Aerodynamic Gradients Using Three Methods," HSR Airframe Technical Review, Los Angeles, CA, Feb. 1998.

Table 1. Derivatives of V with respect to tws ($\overline{tws}$).

Wing Section	CFL3D 5.0		CFL3D 4.1	
	ADIFOR Forward Mode	ADJIFOR Iterated Reverse Mode	ADIFOR Forward Mode	ADJIFOR Iterated Reverse Mode
1	-8.9783175395046E-02	-8.9783304818605E-02	-9.3596682272919D-02	-9.3596521007446D-02
2	-0.13777493172478	-0.13777513263480	-0.14369275653618	-0.14369243579322
3	-0.14033316309455	-0.14033340058874	-0.14565042805518	-0.14565008124860
4	-0.14268876834394	-0.14268902700298	-0.14780920646490	-0.14780883618091
5	-0.14484174747294	-0.14484201187753	-0.15016909176536	-0.15016870059014
6	-0.19469374919372	-0.19469415020925	-0.20168090120663	-0.20168033692085
7	-0.14544334149336	-0.14544361023223	-0.15104420505533	-0.15104381205916
8	-0.13203643206894	-0.13203667493141	-0.13700349641320	-0.13700314053909
9	-0.13562460240379	-0.13562481525497	0.14102086061985	-0.14102054023426
10	-0.15620785249790	-0.15620803120294	-0.16309629767528	-0.16309601114465
11	-2.2707098520427E-02	-2.2707403366190E-02	-8.2278444748449D-02	-8.2278045874506D-02

Table 2. Comparison of timings for baseline function evaluation and derivative computation using one-sided finite differences, forward mode, and the iterated reverse mode with a single processor, CFL3D 5.0.

Machine		Timings (seconds)	Ratio to Function
SUN SMP	Function	111	1
	One-sided FD	987E+1	89
	Forward Mode	324E+2	292
	Iterated Reverse Mode	810	7.3
IBM SMP	Function	132	1
	One-sided FD	117E+2	89
	Forward Mode	530E+2	402
	Iterated Reverse Mode	146E+1	11.1
SGI SMP	Function	175	1
	One-sided FD	156E+2	89
	Forward Mode	123E+3	701
	Iterated Reverse Mode	375E+1	21.4

Table 3. Summary of timings (in seconds) for baseline function evaluation and the iterated reverse mode for CFL3D 4.1 on the IBM SMP.

Test Case	33×9×9				65×17×17			
# zones	1	2	4	8	1	2	4	8
Function Time	141	107	88	71	1385	1070	682	336
Iterated Reverse Mode Time	1447	870	625	426	11697	7606	4634	2568
Ratio of Iterated Reverse Mode to Function Time	10.2	8.13	7.10	6.00	8.45	7.10	6.79	7.64

Table 4. Summary of storage requirements per compute node for baseline function evaluation and the iterated reverse mode for CFL3D 4.1 on the IBM SMP.

Test Case	33×9×9				65×17×17			
# zones	1	2	4	8	1	2	4	8
# pts	2673	1377	765	425	18785	9537	5049	2673
Function Memory	9.7M	9.6M	9.9M	10M	14M	13M	13M	14M
Iterated Reverse Mode Dynamic Memory	21M	11M	6M	3.4M	150M	75M	40M	21M
Iterated Reverse Mode Static Memory	17M	17M	18M	18.6M	20M	23M	25M	27M
Total Iterated Reverse Mode Memory	38M	28M	24M	22M	170M	98M	65M	48M
Iterated Reverse Mode Disk	84M	44M	25M	14M	601M	304M	165M	90M
Iterated Reverse Mode Dynamic Memory (bytes/pt)	7776	7745	7875	7965	7885	7824	7923	7950
Iterated Reverse Mode Disk (bytes/pt)	31598	31683	32518	33918	31969	31907	32610	33659

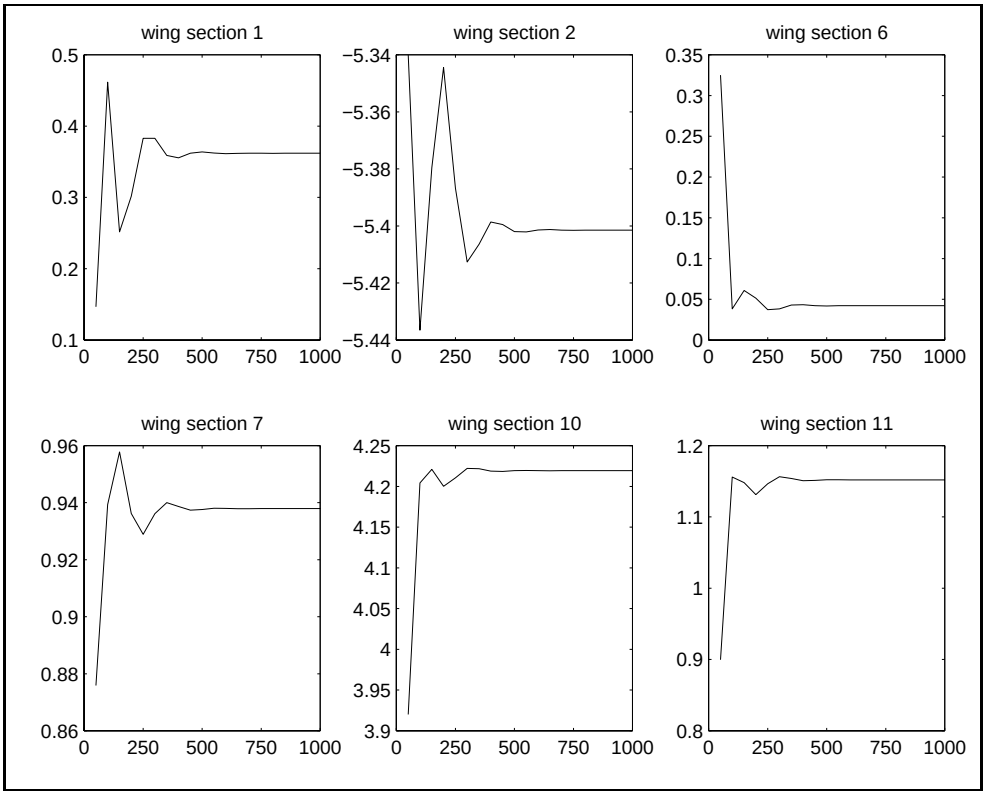

Fig. 1. Iterated reverse mode AD convergence for the derivatives of V with respect to xle ($\overline{xle}$), CFL3D 5.0.